Learning Thin Junction Trees via Graph Cuts

Dafna Shahaf   Anton Chechetka   Carlos Guestrin
Carnegie Mellon University
{dshahaf, antonc, guestrin}@cs.cmu.edu

Abstract

Structure learning algorithms usually focus on the compactness of the learned model. However, for general compact models, both exact and approximate inference are still NP-hard. Therefore, the focus only on compactness leads to learning models that require approximate inference techniques, thus reducing their prediction quality. In this paper, we propose a method for learning an attractive class of models: bounded-treewidth junction trees, which permit both compact representation of probability distributions and efficient exact inference.

Using Bethe approximation of the likelihood, we transform the problem of finding a good junction tree separator into a minimum cut problem on a weighted graph. Using the graph cut intuition, we present an efficient algorithm with theoretical guarantees for finding good separators, which we recursively apply to obtain a thin junction tree. Our extensive empirical evaluation demonstrates the benefit of applying exact inference using our models to answer queries. We also extend our technique to learning low tree-width conditional random fields, and demonstrate significant improvements over state of the art block-L1 regularization techniques.

1   Introduction

Structure learning algorithms have traditionally focused on learning *compact* probabilistic graphical models (PGMs). However, compactness by itself is not sufficient to guarantee that the learned models are useful in practice. PGMs are mainly used for inference, but even approximate inference is NP-hard in compact graphical models. Therefore, one usually has to resort to approximate inference methods, which can yield unreliable results.

Luckily, there exist exact inference algorithms with complexity exponential only in *treewidth* of the model, so for models with low treewidth, exact inference can be performed efficiently. In this paper, instead of taking the common approach of learning complicated models that fit the data well and running approximate inference on them, we will learn simpler models (specifically, low-treewidth junction trees (Bach and Jordan [2002])) and use exact inference.

There has been a substantial amount of work on structure learning (Chow and Liu [1968], Karger and Srebro [2001], Bach and Jordan [2002]). For n variables and desired treewidth k, existing methods for learning low-treewidth structures either have complexity of at least $O(n^k)$ (Chechetka and Guestrin [2008]), or take local, myopic steps (Bach and Jordan [2002]). Unlike previous methods, our algorithm takes steps that are based on a more global criterion, while having complexity polynomial not only in n, but also in k. In addition to attractive asymptotic complexity, our approach is also very fast in practice.

In this paper, we show that under Bethe approximation of likelihood (Yedidia et al. [2000]), finding the optimal JT separator is equivalent to finding a minimum-weight cut in a weighted graph for a certain metric. Based on the graph cut formulation, we present an efficient principled algorithm for finding good separators, which we recursively apply to obtain a thin junction tree. We evaluate our method on different synthetic and real-world datasets and show it to be faster than existing algorithms while in most cases learning models of similar or better quality.

As our algorithm only requires a weighted graph as input, it can be easily adapted to criteria other than Bethe approximation. As an example, in this paper we extend our approach to learning low-treewidth conditional random fields (Lafferty et al. [2001]) and demonstrate significant empirical improvements over the state of the art techniques.

The main contributions of this paper are:

- Formulating the problem of structure learning via graph cuts.
- A fast randomized algorithm with theoretical guarantees for finding good separators and combining them.
- Extending the algorithm to conditional random fields.

2 Learning Junction Trees

In this paper, we learn a certain class of PGMs, namely JTs with low treewidth, for which exact inference can be performed efficiently. In this section, we briefly review these models (for more details, see Cowell et al. [2003]).
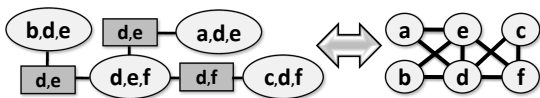

Figure 1: Left: a junction tree. Ellipsoids are cliques, rectangles are separators. Right: the corresponding Markov Network.

Let $\mathbb{C} = \{C_1, ..., C_m\}$ be a collection of subsets of V. Elements of $\mathbb{C}$ are called cliques. Let T be a set of edges connecting pairs of cliques such that $(T, \mathbb{C})$ is a tree.

Definition 2.1 (Junction Tree (JT)) $(T, \mathbb{C})$ *is a junction tree iff it satisfies the running intersection property:*

$\forall C_i, C_j \in \mathbb{C}$ *and* $\forall C_k$ *on the (unique) simple path between* C_i *and* C_j *,* $x \in C_i \cap C_j \Rightarrow x \in C_k$*.*

A set $S_{ij} \equiv C_i \cap C_j$ is called the *separator* corresponding to an edge $(i - j) \in T$. The size of a largest clique in a junction tree minus one is called the *treewidth*. A k-JT is a junction tree of treewidth at most k. For example, the tree on the left Figure 1 is a 2-JT. On the right is the corresponding Markov network: there is an edge between every two vertices that appear together in a JT clique.

A distribution $P(V)$ is representable using $(T, \mathbb{C})$ if instantiating all variables in any separator S_{ij} renders the variables on different sides of S_{ij} independent. In this case, a projection of P on $(T, \mathbb{C})$, defined as $P_{(T, \mathbb{C})} = (\prod_{C_i \in \mathbb{C}} P(C_i))/(\prod_{(i-j) \in T} P(S_{ij}))$ is equal to P itself. Otherwise, projection $P_{(T, \mathbb{C})}$ provides the best possible approximation (in a Kullback−Leibler sense) for a JT with structure $(T, \mathbb{C})$.

In this paper, we address the following problem: given data D and an integer k, we treat each datapoint as a complete instantiation of the random variables V and seek to find a good tractable approximation of $P(V)$. Specifically, we try to find a junction tree of treewidth at most k that maximizes the log likelihood of the data, $\max_{(T, \mathbb{C})} \log P(D|(T, \mathbb{C}))$.

3 Maximizing Bethe Approximation to Likelihood

We want to efficiently find a low-treewidth model with high log-likelihood. Maximizing a model's likelihood is equivalent to minimizing its empirical entropy $H(V)$. Given a JT, we can compute its entropy efficiently: since entropy decomposes as a sum over clique entropies minus separator entropies, we only need to compute entropy of limited-size sets. Methods for structure learning (e.g., Chechetka and Guestrin [2008]) typically need to explore a large number of candidate cliques and separators, and thus are forced to compute many such entropies, potentially over all possible $O(n^{k+1})$ sets. To avoid such a costly dependence on the treewidth k, we propose a scoring method based on a *Bethe free energy* approximation of the entropy Yedidia et al. [2000].

Definition 3.1 (Bethe Entropy) *Let* $G = (V, E)$ *be an undirected Markov network. The Bethe entropy,* H_β*, is the sum of edge entropies, minus the entropies of over-counted intersections.*

$$H_\beta(V, E) = \sum_{(u,v) \in E} H(u, v) - \sum_{v \in V} (\deg(v) - 1) H(v)$$

A JT can be represented by a Markov net, by creating an edge between every pair of variables that appear in the same JT clique, (e.g., Figure 1). We will thus focus on finding a JT with low H_β.

Claim 3.2 *Minimizing* H_β *is equivalent to maximizing* $\sum_{(i,j) \in E} I(V_i, V_j)$*.*

Proof sketch: $\sum_{(i,j) \in E} I(V_i, V_j) = -H_\beta(V, E) + \sum_i H(V_i)$, and $\sum_i H(V_i)$ is constant.

We need a JT with edges that are as strong as possible, in terms of mutual information. One natural way to get a high-likelihood, low-treewidth model is to start from a dense model and remove weak edges (weighted by the mutual information) until the treewidth is low enough. Motivated by this, we reformulate the problem from Section 2:

Definition 3.3 (Edge removal for low-treewidth learning problem)
Input: $G = (V, E)$*, an undirected graph, weight function* $w : E \to \mathbb{R}^+$ *and an integer* k*.*
Goal: remove the lowest-weight set of edges such that the resulting graph has treewidth $\leq k$*.*

In our settings, edge weights come from mutual-information estimations. Weights could also come from L1 regularization, or other methods that estimate pairwise dependence. Later we learn conditional random fields, and edges are weighted by conditional mutual information.

4 Finding a Low-Weight Separator

Given a dense, weighted graph, our goal is to remove weak edges until the treewidth is low enough. We achieve this goal by identifying potentially good JT separators, one by one. Later we can recover the tree from the list of separators and their corresponding connected components. We notice that if S is a separator in a JT, it cuts the graph into two connected components.

Definition 4.1 (Cut) *A cut* $(A, B|S)$ *is a partition of* V *into disjoint subsets* $V = A \cup B \cup S$, *such that* $A, B \neq \emptyset$. S *is called a* separator.

We want to find a cut such that removing separator S renders A and B almost independent. By the Bethe approximation, we want the weight of the cut, $\sum_{a \in A, b \in B} w(a, b)$, to be low. Cut edges are the edges we remove, and thus their weight corresponds to loss in Bethe entropy. Figure 2
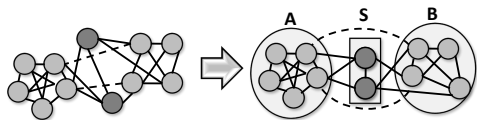

Figure 2: Left: A mutual-information graph. Dashed edges have lower weight. Right: A partition with a separator of size 2, s.t. edges between A and B are as weak as possible. Note, the actual JT cliques are $A \cup S, B \cup S$.

illustrates this idea. A weighted graph G is on the left; dashed edges are weaker (have low weight). There is no separator of size 2 that cuts the G into independent components; however, there is a separator S that partitions the graph into components that have only weak edges between them, A and B. Note that we only pay for edges from A to B directly. The edges that go through S might be strong, but we do not pay for them. In our approach, we will use S as a separator in our JT and recurse on both sides, $S \cup A$ and $S \cup B$. In the following, we formalize this intuition.

Definition 4.2 (min k-Node-Removal Edge Cut Problem)
Input: an undirected graph $G = (V, E)$, *a non-negative weight function* $w : E \to \mathbb{R}^+$ *and an integer* k.
Goal: find a separator set S *of at most* k *nodes, and a partition* (A, B) *of the remaining variables, such that the weight of* cut $(A, B|S) = \sum_{a \in A, b \in B} w_{ab}$, *is minimized.*

4.1 Solution via Integer Programming

In this section we encode the problem as an integer program. We later relax it to a linear program and round the LP solution. We want a low-cost cut $(A, B|S)$, with $|S| \leq k$. Suppose we know in advance some $a \in A, b \in B$. We can formulate the problem as an integer program:
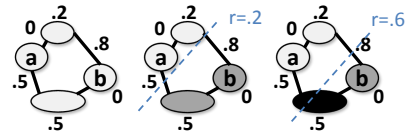

Figure 3: Rounding demonstrated. Left: LP solution. Numbers denote edge/node internal distance. Note that $d_s = 0, d_t = 1$. Two cuts are shown, for different radii. For $\rho = 0.6$, there is one node in the separator.

$$
\begin{aligned}
min \quad & \sum_{e \in E} w_e \cdot c_e \\
s.t. \quad & \sum_{v \in V} s_v \leq k & (1) \\
& s_a = 0, s_b = 0 & (2) \\
& d_a = 0, d_b = 1 & (3) \\
& d_u \leq d_v + s_v + c_{(v,u)} & (4) \\
& c_e, s_v, d_v \in \{0, 1\} & (5)
\end{aligned}
$$

$c_e = 1$ indicates that we drop edge e (i.e., e is a cut-edge). The objective, $\sum_{e \in E} w_e \cdot c_e$, is the cost of dropped edges. $s_v = 1$ indicates that node v is in the separator S. Constraints $(1), (2)$ assert that $|S| \leq k$ and that $a, b \notin S$. In addition, the IP defines an indicator variable $d_v = 1$ if $v \in B$ for every node v. Constraint (3) asserts that $a \notin B, b \in B$. Finally, constraint (4) is a transition constraint: for every edge (u, v), if $u \in B$ and $v \notin B$, either $v \in S$ or we remove the edge (u, v). In other words, on every path from A to B there is either a node from S, or a cut-edge which we remove. An integer solution to this problem specifies a cut $(A, B|S)$, such that $|S| \leq k$.

Recall that we assumed knowing in advance some $a \in A, b \in B$. However, this is not the case. The naive way would consider running the IP for all $\binom{n}{2}$ pairs of vertices; instead, we note that we can arbitrarily pick one vertex to be a, and test the other $n - 1$ vertices as b. In order to guarantee a in not in the separator, we may need to pick $k + 1$ potential sources, for a total of $O(nk)$ checks (see Alg. 1). In practice, we can often stop after a smaller number of checks, especially for sparse graphs, using the following property: the IP objective is non-negative, so if the IP finds a 0-weight cut, it is an optimal solution, and cannot be improved.

4.2 LP Relaxation and Rounding

Integer programming is NP-hard. Instead, we solve the linear programming (LP) relaxation, obtained by replacing integrality constraints (5) by range constraints $0 \leq c_e, s_v, d_v \leq 1$. LPs are solved in polytime, but may return a fractional solution. The next step is to derive a separator from the optimal LP solution (Alg. 2). We describe our randomized rounding procedure as follows:

The LP returns $c_e, s_v, d_v \in [0, 1]$. We interpret those as distances: c_e is the length of an edge, and s_v is the length of a node (some nodes are bigger than others). The length

---

Algorithm 1: $findSeparator(G, w, k)$

Input: $G = (V, E)$: undirected graph, w: weight function, k: max separator size.
Output: $(A, B|S)$ cut of V
1 **if** $|V| \leq k + 1$ **then return** $(V, \phi|\phi)$; // found a small clique, no need to separate
2 **for** $i = 1$ to $k + 1$ **do**
3   Pick $a \in V$
4   **forall** $b \in V, b \neq a$ **do**
       minimize $\quad \sum_{e \in E} w_e \cdot c_e$
       subject to $\quad 0 \leq c_e, s_v, d_v \leq 1$
               $\sum_{v \in V} s_v \leq k$
               $d_a = 0, d_b = 1, s_a = 0, s_b = 0$
5                $d_u \leq d_v + s_v + c_{(v,u)}$
6 Pick a^*, b^* with the minimal LP value
7 **return** $roundCut(G, w, k, \langle a^*, b^*, c_e^*, s_v^*, d_v^* \rangle)$

Algorithm 2: $roundCut(G, w, k, \langle a, b, c_e, s_v, d_v \rangle)$

Input: $G = (V, E)$ graph, w weight, k max separator size, the LP solution for a, b
Output: $(A, B|S)$ cut of V
1 distances= $uniqueSet(\{d_v\} \cup \{d_v + s_v\})$
2 distances= distances $\setminus \{d_b\}$
3 **forall** $\rho \in$ distances ;   // derandomize (Section 4.2)
4 **do**
5     $A = \{v \in V \mid d_v + s_v \leq \rho\}$;   // Find best k-node cut for A
6     $B' = V \setminus \{A \cup b\}$; $val_{b'} = \sum_{a' \in A} w_{(a',b')} \;\; \forall b' \in B'$
7   Pick $S \subseteq B'$ with highest $\sum_{s \in S} val_s, |S| \leq k$
8     $B = V \setminus \{A \cup S\}$
9   weight$(A, B|S) = \sum_{a' \in A, b' \in B} w_{(a',b')}$
10     $checkValid$(G,w,S) ; // when called recursively, need to check if S can be added to current JT (Section 5)
11 **if** no finite-weight valid $(A, B|S)$ exists **then return** some ∞-cost cut
12 **else return** the minimum-weight valid $(A, B|S)$

---

of a path from u to v is the sum of its edge lengths and internal node lengths. Under this interpretation, d_v is the distance from vertex a to vertex v (i.e., the minimal path length). Constraints $(3), (4)$ thus assert that the distance from a to itself is 0, the distance from a to b is 1.

We pick a radius $\rho \in [0, 1)$ uniformly at random, and partition the graph into sets (A, B, S) s.t. $a \in A, b \in B$. Let $A = \{v \mid d_v + s_v \leq \rho\}$, the nodes completely inside the ρ-ball centered at s. $S = \{v \mid d_v < \rho < d_v + s_v\}$ are the nodes cut by ρ. B are the rest, $\rho \leq d_v$ (see Figure 3).

Proposition 4.3 *The expected cost of roundCut is at most OPT_{LP}, the optimal objective value of the LP relaxation: $E_\rho(weight\ of\ cut) \leq \sum w_e \cdot c_e = OPT_{LP} \leq OPT_{IP}$, where OPT_{IP} is the optimal value of the integer program. The expected size of S is at most k: $E_\rho(|S|) \leq \sum s_u = k$*

Proof sketch: $P(e\ in\ cut) \leq ((\rho + c_e) - \rho) = c_e$. $P(u \in S) \leq ((\rho + s_u) - \rho) = s_u$. Proposition 4.3 provides a guarantee in expectation. However, for each value

of ρ typically the cut cost would be higher, or S would be too big, so we need to find a Pareto choice. Fortunately, we have a derandomization scheme that provides the entire approximate Pareto frontier. We note that although ρ can take any value in $[0, 1)$, there are only $2n$ critical points: transitions happen only around $\{d_v\}, \{d_v + s_v\}$. By the probabilistic method, if we try each one of those radii we will find a solution with either $|S| \leq k$, or value $\leq OPT_{LP}$. This frontier can allow us to regularize, if we wanted to and pick a smaller (or even larger) S. However, in this paper, we are only interested in $|S| \leq k$. Therefore, we only explore that part of the frontier, and pick the weakest cut.

A variant of this derandomization scheme (Alg. 2) guarantees that $|S| \leq k$. Given a radius ρ, compute A as before. By definition, S is a subset of A's neighbours. Each neighbour b can either belong to S or to B. If $b \in B$, we pay for all the edges between b and A. If $b \in S$, we do not need to pay for these edges. We compute the weight of edges between A and b, val_b, and pick up to k nodes greedily for S. This results in the optimal separator $|S| \leq k$ for this A.

Claim 4.4 *The greedy derandomization scheme is guaranteed to find a cut that does not weight more than any cut $(A_\rho, B_\rho|S_\rho)$ generated by some sampled ρ s.t. $|S_\rho| \leq k$.*

5 Recursive Procedure

Now we have a procedure for finding a partition of the graph, where both sides A, B are connected by weak edges. We can use S as a separator in our JT, and recurse on both sides of the partition – $A \cup S$, $B \cup S$, stopping when the set is of size $k + 1$. However, this simple approach will not work. The cliques and separators generated might not guarantee the existence of a k-JT: if we add an edge between every two variables which appear in the same clique, the Markov network generated might have a treewidth higher than k. By definition, this means that the cliques and separators cannot be combined to form a k-JT.

See Figure 4 for an example. We try to find a 2-JT over 10 variables: 1,..,4,a,..,f. Suppose in the first 6 steps the algorithm chooses a unique pair in $\{1, 2, 3, 4\}$ for S, and splits one of $a,...,f$ from the rest. After six steps, the clique $\{1, 2, 3, 4\}$ is too big, but cannot be split anymore; if we split any two variables that have previously appeared in a separator, we lose the running intersection property.

Recall the problem definition (4.2). We are given a weighted clique; our goal is to remove a subset of the edges, such that the resulting graph has a low treewidth, and the weight of the removed edges is minimal. At the beginning, all of the edges are *undecided*. When we choose a cut $(A, B|S)$, we commit to removing edges between A and B, and keeping all the internal edges of S.

In other words, the set of edges E can be partitioned into three subsets of edges, $E_+ \cup E_- \cup E_?$: edges we commit to keep, edges we remove, and edges that are undecided.
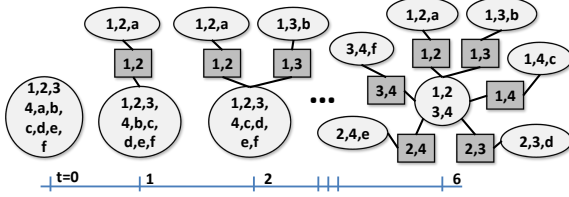
Figure 4: A sequence of six separation steps (left to right) resulting in cliques that cannot be made into a JT of treewidth 2.

Our algorithm successively picks cuts in the graph. We start with $E_? = E$, and as the algorithm proceeds it moves edges from $E_?$ into E_+, E_- until eventually $E_? = \emptyset$.

In general, there are two situations that might prevent us from getting a k-JT: We can end up with $E_+ \cap E_- \neq \emptyset$, i.e. a separator formed in the earlier part of the recursion is cut in a latter step, which violates the running intersection property (**Sep-RIP**). Alternatively, the treewidth of the graph induced by E_+ may be larger than k (**High-TW**).

In order to ensure the running intersection property for separators (**Sep-RIP**), after choosing a separator we make it into an ∞-weighted clique (see $findJT$). This way, vertices of S might be used in other separators, but splitting them would cost ∞. Therefore, any finite-weight cut does not split previous separators; in other words, $E_+ \cap E_- = \emptyset$.

Claim 5.1 *After choosing a separator S for the JT, S will not be split in latter iterations.*

In order to solve the **High-TW** problem, we need to look at the graph induced by E_+. By construction, those are exactly the ∞-edges.

Claim 5.2 *Let $G_\infty = (V, E')$, such that $E' = \{e \in E \mid w_e = \infty\}$ (the ∞-edges subgraph, E_+). If the treewidth of G_∞ is $\leq k$ throughout the (recursive) computation, $findJT$ returns a k-JT.*

Note that we cannot allow the treewidth of G_∞ to exceed k at any stage of the algorithm. As E_+ can only grow, the treewidth can only increase from that point.

Claim 5.3 *If the treewidth of G_∞ is $\leq k$ and $E_? \neq \emptyset$, there exists a finite-weight cut $(A, B|S)$ which keeps the treewidth of G_∞ at most k and reduces the size of $E_?$.*

If the treewidth of G_∞ is $\leq k$, there is a triangulation where all separators are of size $\leq k$. Choosing S from this triangulation maintains the treewidth of G_∞ at most k. For example, refer to Figure 4 again: after picking the last separator, $\{3, 4\}$, G_∞ had treewidth 4. Looking at the triangulation in the previous step, we see cliques $\{1, 2, 3\}, \{1, 2, 4\}$ and separator $\{1, 2\}$. And indeed, picking $S = \{1, 2\}$ and splitting 3 from 4 leads to a valid 2-JT.

In order to find the cut guaranteed by Claim 5.3, we use the min-fill heuristics to bound the treewidth of G_∞ from

---

Algorithm 3: $findJT(G, w, k)$

Input: $G = (V, E)$: undirected graph, w: weight function, k: max separator size.
Output: a list of cliques that can form a k-JT

1 **if** G has several connected components **then** recurse on each, **return**
2 **if** $|V| \leq k + 1$ **then** record V in the JT. **return**
3 $(A, B, S) = findSeparator(G, w, k)$
4 **if** S *has ∞-weight* ; // no finite separator found
5 **then**
6   Maintain Claim 5.2: Find valid S using min-fill heuristic
7   run min-cuts to find corresponding A and B
8 Change G, w such that S is a clique with ∞-weighted edges
9 $findJT(G[A \cup S], w, k, S)$; // got a valid S, recurse
10 $findJT(G[B \cup S], w, k, S)$

Algorithm 4: $checkValid(G, w, S)$

Input: $G = (V, E)$: undirected graph, w: weight function, k: max separator size, S: potential separator.
Output: Does there still exist a k-JT if we add S?
1 Change G, w such that S is a clique with ∞-weighted edges
2 $G' := G_\infty = (V, E')$, $E' = \{e \in E \mid w_e = \infty\}$
3 $k' =$Use min-fill heuristic to bound tree-width of G'
4 **if** $k' \leq k$, **return true else return false**

---

above (see Alg. 4). We can use any other triangulation method insead of min-fill. If the LP separator violates **High-TW**, we drop it and look for another one (e.g. constraining the LP to use at most $|S| - 1$ of S's vertices, $\sum_{v \in S} s_v \leq |S| - 1$).

If we were to solve the IP, $findJT$ would be guaranteed to find a finite weight separator at every step, and thus a k-JT. However, the rounding may not find a finite cut, even if one exists. If $findSeparator$ returns an infinite cut, we discard it and instead use a separator we got from min-fill in the previous iteration. A, B are obtained using the min-cuts algorithm. This situation seems to arise very rarely in practice.

Proposition 5.4 *$findJT$ is guaranteed to find a k-JT.*

5.1 Objective Function

Our objective function in Section 4 minimizes the total weight of removed edges. Since we only handled finding a single cut, all of the edges not removed were assumed to stay in the graph. However, this is no longer the case: the undecided edges $E_?$ may be removed later. For this reason, we suggest changing the objective function, taking into account both the edges we remove and the edges we keep:

$$\min \left(\sum_{e \in E} w_e \cdot c_e - \sum_{e \in E. w_e < \infty} w_e \cdot insep_e \right)$$

The first component of the sum is the previous objective: w_e is the edge weight and c_e indicates a cut-edge. $insep_e$ are new variables indicating that edge e is an internal edge

of S. We achieve this by adding constraint (6): $insep_e \leq s_u, insep_e \leq s_v$. The edges with $insep_e = 1$ are those with both ends in S, i.e. exactly the edges which are added to E_+. We need to be careful not to double count edges that are already in E_+, so we sum over finite-weight edges only. The second component of the new objective is therefore the total weight of newly added edges (alternatively, we can also add a tradeoff parameter λ between the sums).

This objective still minimizes the weight of E_- (the sum of weights of E_- and E_+ is fixed). By taking into account the edges of the separator, the new objective also has the advantage of picking strongly-connected separators.

5.2 Running Time and Complexity

Claim 5.5 *findJT is polynomial in n and k.*

The size of the graphs G passed to $findSeparator$ decreases with every iteration; therefore, the number of recursive calls is linear in n. $findSeparator$ calls the LP solver $k+1$ times. Each LP involves $O(n^2)$ variables and constraints, and thus is solved in polytime. Rounding the cut is again polynomial in n (we consider $2n$ radii). Thus, the total running time is polynomial in n and k.

We note that rounding is a fast process in practice: very often the LP solver returns an integer solution, or one with few distinct values, so the number of radii we need to consider is much less than $2n$. Also, note the LP solution is always non-negative. When the LP solver returns a zero-cost solution, we do not need to continue looking for more pairs of vertices a, b to separate.

5.3 A Note on Regularization

Regularization is often used for estimation of sparse models in order to avoid overfitting. We considered several types of regularization. As described, $findJT$ learns maximal JTs (all separators are of size k). However, smaller cliques and separators lead to simpler models. Instead of stopping when clique size reaches $k+1$, we can let $findSeparator$ split the clique further, if the cost is small, as defined by some threshold. A drawback of this techniques is the assumption that each node has the same amount of regularization (in our experiments, this constant was set by cross validation).

Secondly, we define a multiplicative-ratio threshold: after computing the best k-separator S, pick the smallest separator that costs at most $1 + \epsilon$ times the cost of S.

Finally, we can restrict our models further by reducing the number of parameters. Instead of learning full clique potentials for the resulting JT, we can assume, e.g., a pairwise-Markov structure and learn its parameters. Regularization reduced overfitting, especially on small datasets.

6 Experimental Results

We have used $findJT$ to learn various low-treewidth graphical models, from synthetic and real-world datasets.

6.1 Utility of Approximation

Approximate inference is a common solution when the treewidth is too large. In contrast, our approach in this paper is to perform *exact* inference on an *approximate* model. In this section we compare these approaches.

We have generated several artificial networks with a relatively-large treewidth (between 10 and 30). The treewidth was picked so that exact inference on the model takes a long time, but is still feasible. We compared $findJT$ to a state-of-the-art structure learner for compact BN (order-based search, OBS). First, we computed log likelihood for corresponding test sets (Figure 5a). Our algorithm did slightly worse, but comparable to OBS. This is not a surprise, since OBS directly tries to minimize likelihood. In addition, it does not try to learn a thin junction tree, and thus has less constraints (indeed, the models that OBS learned had high treewidth).

The real goal of structure learning is the ability to run inference efficiently and accurately. Therefore, as another way to evaluate the quality of our learned models, we have compared the quality of inference. We fixed a random subset of variables as evidence, and conditioned the models on them. We ran (1) exact inference on the thin models we learned, and (2) fast approximate inference algorithm (residual loopy belief-propagation) on the OBS model. We compared those to the ground truth, exact inference on the real model P_t. For each variable v we computed the KL-Divergence from v's ground truth conditional to the corresponding conditional in the approximate model P_a. We averaged over the variables:

$$Score_a = \frac{1}{n} \sum_v D_{KL}(P_t(v|evidence)||P_a(v|evidence))$$

We let loopy OBS+BP run for the same time our algorithm took, which was usually a few minutes (note that our implementation is in Matlab, while loopy BP is in C++). Refer to Figure 5ab for the results. Each point represents a single network: the x axis is the score of our algorithm, and y – of approximate inference. Most points are above the $x = y$ line, or very close to it, meaning our algorithm was closer to the ground truth.

6.2 Classification Accuracy

In addition to the synthetic datasets, we have tested $findJT$ on several real-world domains. As baselines for comparison, we used LPACJT (Chechetka and Guestrin [2008]), and algorithms of Karger and Srebro [2001] (denoted KS) and Teyssier and Koller [2005] (denoted OBS), all reproduced from Chechetka and Guestrin [2008]. The necessary entropies were cached in advance. KS and LPACJT learn
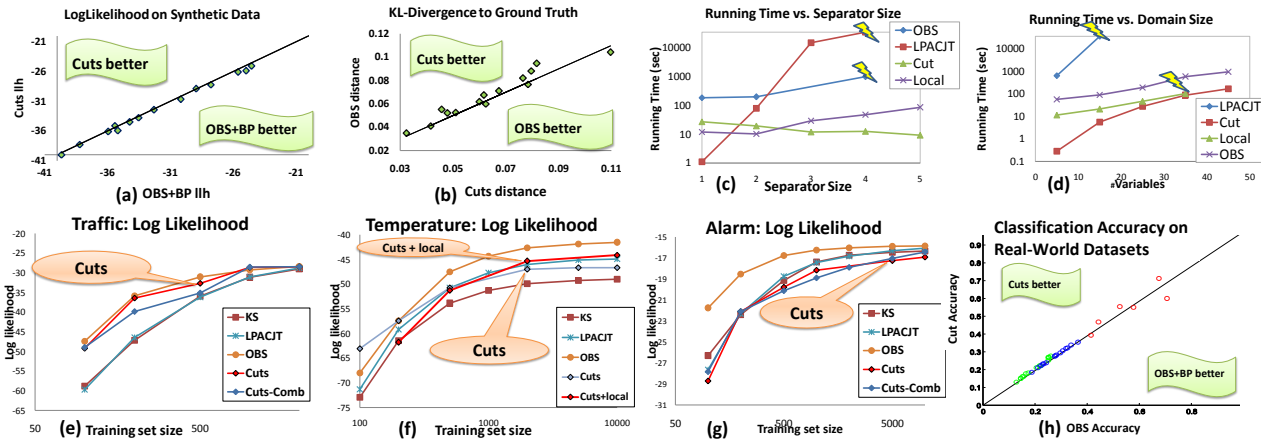
Figure 5: (a),(b): Likelihood and KL-divergence for our algorithm vs. OBS+loopy BP on synthetic data. lh experiments. (c),(d): runtime comparison, vs. separator size and domain size. Lightening signs indicate sudden death of the algorithm, usually running out of memory. (e)-(g): llh experiments on real data. Cuts+Local is running local search from the structure found by the Cut algorithm. Cuts+Comb is the algorithm with the combined objective of Section 5.1. (h) classification accuracies, same experiment.

k-JTs, and OBS learns compact Bayes nets with at most k parents for every variable. The datasets we used were:

Traffic measured every 5 minutes in 32 locations in California for a month Krause and Guestrin [2005]. Discretized into 4 bins, learned models of treewidth 3.

Temperature 2-month deployment of 54 sensors (15K datapoints) Deshpande et al. [2004]. Discretized into 4 bins, learned models of treewidth 2. The sensor locations have an ∞-like shape, so learning a thin JT is hard.

Alarm Discrete data sampled from a known Bayesian network Beinlich et al. [1988], treewidth 4. Due to computational limitations of other methods, we only learned treewidth 3 models.

Our method ran significantly faster than others, taking no more than a few minutes. Algorithms of $O(n^k)$ complexity could not handle separators larger than 3 on the full datasets, or took hours to complete. Refer to Figure 5cd for running-time comparisons between the algorithms Another interesting point illustrated in Figure 5 is that our algorithm sometimes takes less time on larger k's: the number of LP calls increases per iteration of $findSeparator$, but the number of iterations is smaller.

We started by comparing log-likelihoods. The results are shown in Figure 5efg: $findJT$ does well on Traffic, but not as well as OBS on Temperature and Alarm. This is reasonable, because OBS learns compact BNs. BNs allow computing likelihood exactly, so the structure may look very good before inference. Like before, we tested classification accuracy. The real structure of Traffic and Temperature is unknown, so we had no ground truth to compare to. Instead, we fixed a random subset of variables as evidence, E; for each test point x^j, we computed the MPA, given the evidence variables $x^j[E]$. The accuracy is the ratio of variables we classified correctly. We compared the

classification accuracies of our model and the OBS model. The results are shown in Figure 5h. Despite the likelihood results, our accuracy is comparable or better in most cases.

7 Structure Learning for Conditional Random Fields

CRFs Lafferty et al. [2001] are undirected graphical models that compactly represent conditional distributions, $P(y \mid x)$ (x: observations, y: labels). Learning CRFs is a harder than learning Bayes Nets; even scoring a structure requires inference. Therefore, most work on CRFs assumes a known structure, often a linear chain or a 2D lattice. There has been very little work on learning CRF's topology from data. A lot of the previous work focuses on special cases, or different objectives. A notable exception is Schmidt et al. [2008], who proposed a method for jointly learning the (sparse) structure and parameters of CRFs, formulating these tasks as a convex optimization problem. They considered block-L1 regularization and found the global minimum of the resulting pseudo-likelihood objective. We ex-
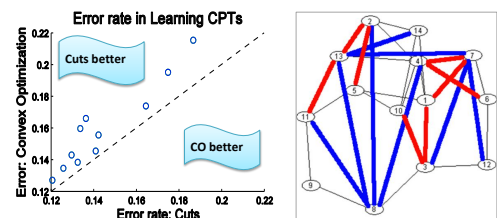

Figure 6: Left: error rate in CRF learning. Right: a learned model. Blue edges appear only in the learned model (due to triangulation), red – only in the real model.

tended our algorithm to CRFs. We assume potentials are of the form $f(Y_i, Y_j, X)$. Initial edge-weights are conditional mutual information, evaluated by logistic regression

for every pair of Y's. In order to compare our structure learning to Schmidt et al. [2008], we have generated several synthetic CRFs, whose size ranges from 8 to 100. To obtain a fair comparison, we used their parameter-learning code on our structures as well. That is, we ran Schmidt's algorithm twice: once for learning a structure and parameters, and one with a fixed structure that we learned. Figure 6 compares the error rate of the approaches. Our algorithm wins for all structures, although it does not try to minimize that error measure. One of those structures learned is also shown in Figure 6. Edges are false positive (due to triangulation), false negative, or true.

8 Relation to prior work

Except for learning the most likely JT of treewidth 1, which can be solved in $O(n^3)$ time (Chow and Liu [1968]), the problem of learning an optimal junction tree of treewidth at most k is NP-complete for most formulations Karger and Srebro [2001]. It is thus not surprising that the existing algorithms for learning limited-treewidth JTs that provide any theoretical guarantees on the quality of the solution (Karger and Srebro [2001], Chechetka and Guestrin [2008]) have complexity at least $O(n^k)$ and are usually not practical for $k > 3$. Other popular approaches are based on local search (Bach and Jordan [2002]), and only guarantee the resulting structure to be a local optimum. Our approach is as fast as the local search-based algorithms and provides an important advantage: our algorithm selects an entire separator at every time step, while most existing local approaches take a more myopic view and only add a single edge.

Recently, Lowd and Domingos [2008] have proposed learning *arithmetic circuits,* a class of models that does not necessarily have low treewidth, but still admit tractable exact inference. Their approach, too, has only local guarantees on the quality of the result.

9 Conclusions and Future Work

Learning high-treewidth models requires using approximate inference methods, which can yield unreliable results. In this paper, we have proposed to learn instead simpler models which allow exact inference. We have proposed a new method for learning bounded-treewidth junction trees, an attractive class of probabilistic graphical models that permit both compact representation and efficient exact inference. By employing Bethe approximation of the model entropy, we have formulated the problem of selecting an optimal junction tree separator in terms of finding a low-weight cut in a graph. We presented a fast randomized algorithm with theoretical guarantees for finding good separators and showed how to combine the separators to obtain a low-treewidth model. We have also extended our approach to learning the structure of discriminative models (CRFs). In addition to theoretical guarantees, we have demonstrated

empirically that our approach produces high-quality results while being faster than the state of the art methods.

An interesting future work direction is extending our techniques to other model formalisms, such as arithmetic circuits (Lowd and Domingos [2008]) and relational networks (Getoor et al. [2001]). We also plan to exploring more general approximations, e.g. Kikuchi (Yedidia et al. [2000]). Our algorithm is currently limited to testing pairwise mutual information; taking higher entropies into account might improve the likelihood. In the future, we plan to enhance our algorithm to handle this case, perhaps by considering hyperedge-cuts. We believe that our general approach can be naturally extended along both of the above directions, which will lead to wider applicability of the algorithm and better quality of the models.

References

F. R. Bach and M. I. Jordan. Thin junction trees. In *NIPS*, 2002.

C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 1968.

D. Karger and N. Srebro. Learning Markov networks: Maximum bounded tree-width graphs. In *SODA*, 2001.

A. Chechetka and C. Guestrin. Efficient principled learning of thin junction trees. In *NIPS*. 2008.

J. Yedidia, W. Freeman, and Y. Weiss. Bethe free energy, Kikuchi approximations, and belief propagation algorithms, 2000.

J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, 2001.

R. Cowell, P. Dawid, S. Lauritzen, and D. Spiegelhalter. *Probabilistic Networks and Expert Systems*. 2003.

M. Teyssier and D. Koller. Ordering-based search: A simple and effective algorithm for learning Bayesian networks. In *UAI*, 2005.

A. Krause and C. Guestrin. Near-optimal nonmyopic value of information in graphical models. In *UAI*, 2005.

A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *VLDB*, 2004.

I. Beinlich, J. Suermondt, M. Chavez, and G. Cooper. The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks. In *European Conference on Artificial Intelligence in Medicine*, 1988.

M. Schmidt, K. Murphy, G. Fung, and R. Rosales. Structure learning in random fields for heart motion abnormality detection. In *CVPR*, 2008.

D. Lowd and P. Domingos. Learning arithmetic circuits. In *UAI*, 2008.

L. Getoor, N. Friedman, D. Koller, and B. Taskar. Learning probabilistic models of relational structure. In *ICML*, 2001.